Restaurant Owner App Description (Project 0, part 1)

Scenario:

> You live in a city that's famous for its food. Year round crowds of people from around the world visit just for a change to try your city's cuisine. As a result, restaurant owners around your city buy MULTIPLE refrigerators in order to store large quantities of food and serve as many customers as possible.
>
> Restaurant owners have difficulty keeping track of their stock and are asking you for your help. They task you with creating a Java application that they can use to organize and keep track of their stored food.

Requirements:

- The application should allow 2 types of accounts to be created: **RESTAURANT** and **HEALTH INSPECTOR**.
  - Different account types will have different options available to it.
- The application should **interact with the account user through the console** using the Scanner class.
- The user should be able to decide which task they'd like to perform. **When the user has completed a task the application's event loop should give the user the option of performing a new task**. "Logout" and/or "Exit" should be an option.
- **All information should be persisted** (saved) using serialization and text files. This step allows you to RESTART your application and STILL see all changes made in the previous run of the application. (Project 0, part 1)
- **You may now ignore the bullet point directly above this bullet point. All information should NOW be persisted (saved) using JDBC and an Oracle DB**. Your database should include/utilize at least 1 stored procedure. (Project 0, part 2)

Restaurant accounts:

- As a restaurant, you should be able to **register a new RESTAURANT account** with a username and password.
- As a restaurant, you should be able to **add refrigerators to your account**. You should be able to add new **or remove** refrigerators at any time. If you delete a fridge the food from that fridge is placed into another fridge (you may not delete your LAST fridge).
  - Fridges may only hold a **maximum of FIVE food items**.
  - Each fridge object should **remember which restaurant purchased it**.
- As a restaurant, you should be able to **deposit food into a fridge**. This will more than likely require the user to use the console to type in a new food item.
- As a restaurant, you should be able to **withdraw food from a fridge**.

- As a restaurant, you should be able to **transfer food between fridges** (restaurants may have 1 or more fridges).
- As a restaurant, you should be able to **give one OR MORE health inspectors permission to a particular fridge**.
    - Since you assign health inspectors to each fridge individually, each fridge may have a different list of health inspectors (even fridges in the same restaurant may have a different list of inspectors).
    - So, a single fridge may be accessed by many inspectors (1 or more) AND a single inspectors may access many fridges (1 or more).
- Basic validation should be done, such as trying to withdraw (or transfer) food that is not present in this specific fridge, making sure any specific fridge doesn't exceed 5 items, etc.

Health Inspector accounts:

- As a health inspector, you should be able to **view ALL fridges you have been given access to**.
    - ONLY the fridges you have been given permission to access, no other fridges.
    - SO, many users can access a single fridge and a single user may have many fridges. This is a many to many relationship….you cannot avoid a junction table.
- As a health inspector, you should be able to **remove the contents from all fridges you have been given access to**.

Part 2 add ons:

- You should use the DAO design pattern for DB connectivity.
- Reasonable test coverage is expected using J-Unit.
    - Folks, i'm not asking for 100% test coverage; but give me at least 5-10 tests to demonstrate that you are capable of using J-Unit. I don't think this is a tall ask.
- Logging should be accomplished using Log4J:
    - All transactions should be logged

**Please take the deadline seriously.

**Do NOT spend too much time stuck on a single blocker without asking a batch-mate for help.

Tips on how to start

If you're having trouble wrapping your head around how to start, here is my preference for starting project 0:

1. Start with your data models. Create all the classes that will be used to simply hold information about each ACCOUNT and each FRIDGE. These objects should do NOTHING but hold information about a single account/fridge.
    a. These classes should basically only have variables, constructors, getters/setters, and a toString.
2. Next, we'll create a class that stores the application's current state. Create a class that is responsible for keeping track of all existing accounts and fridges. It will be THIS class that will perform operations all groups of accounts or fridges.
    a. This class(es) will have methods like "findAllAccounts()" and "findAllFridges()" and "findAccountByUsername(String username)" and etc.
    b. This class will also have a method that is responsible for persisting/saving information. The method should be named appropriately (e.g. "persistAllData()").
3. Next, create a class that will be responsible for interacting with the user. In this class you'll first set up a simple event loop that allows the user to enter different commands to perform different tasks.
    a. For example, the user could enter "withdraw" or "yes" or etc.
    b. I say "simple" because you shouldn't use the data models or application state functionality yet; JUST focus on the event loop, and add the actual operations later.
4. Now, in the event loop you may have different sub menus at different times. Each of these sub menus should be either a DIFFERENT method or a DIFFERENT class (for organizational and readability purposes).
    a. Examples of menus, before you login the menu may be something like "(1) register (2) login (3) exit" but after you login you could have a menu like "(1) add fridge (2) withdraw (3) deposit".
5. Once the menus are done, you can slowly begin to attach functionality to each menu option. Since we have completed our data models and application state class we can just modify our collection of accounts/fridges in some way to perform each menu operation.
6. Finally, after each task/operation has concluded work on persisting that information to the text file. Don't forget that your utility class and persisted information should be in sync; so if a user makes a change both need to be notified.

Note: this is not the only way to start the project but if you don't know how to start you could use this guideline.